Prvi kolokvijum iz Operativnih sistema 1
Odsek za softversko inženjerstvo
Mart 2013.

1. (10 poena)

```
static REG* ioPtr = 0;
static int ioCount = 0;
static int ioCompleted = 0;

void transfer (int count) {
  REG* buffer = new REG[count];

  // I/O 1:
  ioPtr = buffer;
  ioCount = count;
  ioCompleted = 0;
  *io1Ctrl = 1; // Start I/O 1

  // Wait for I/O 1 completion:
  while (!ioCompleted);

  // I/O 2
  ioPtr = buffer;
  ioCount = count;
  *io2Ctrl = 1; // Start I/O 2
  while (ioCount>0) {
    while (!(*io2Status&1)); // busy wait
    *io2Data = *ioPtr++;
    ioCount--;
  }
  *io2Ctrl = 0; // Stop I/O 2

  delete [] buffer;
}

interrupt void io1Interrupt() {
  *ioPtr++ = *io1Data;
  if (--ioCount == 0) {
    ioCompleted = 1;
    *io1Ctrl = 0; // Stop I/O 1
  }
}
```

2. (10 poena)

VA: Segment(2):Page(6):Offset(8); PA: Frame(24):Offset(8).

3. (10 poena) Kada se pri obradi prekida kontekst procesora prepisuje u memoriju, bitno je da u registru SPX bude adresa polja *context* iz PCB-a tekuće niti koja gubi procesor. Isto važi pri svakoj promeni konteksta, pa i pri napuštanju kernel niti. Vrednost samog SPX se ne mora čuvati, jer se jednostavno restaurira dodavanjem pomeraja polja *context* na vrednost adrese PCB-a. Dakle, jedino što je prilikom promene konteksta potrebno uraditi jeste postaviti SPX na vrednost adrese polja *context* niti koja dobija procesor. Tako rutina `sys_call` izgleda ovako:

```
sys_call:   load  spx, [runningKernelThread]
            add   spx, #offsContext
            iret
```

Povratak iz rutine će restaurirati kontekst kernel niti iz strukture *context* PCB-a niti koja je dobila procesor, jer na tu strukturu tada ukazuje SPX. Potpuno analogno izgleda i rutina kojom se kontrola iz kernel niti predaje korisničkoj niti, samo što je umesto `runningKernelThread` upotrebljen `runningUserProcess`.

4. (10 poena)

```
const int N = ...;
int f(int,int);
#include <iostream.h>

struct f_params {
  int i, j, r, pid;
};

f_params params[N*N];

void f_wrapper (int index) {
  params[index].r=f(params[index].i,params[index].j);
}

void main () {
  for (int i=0; i<N; i++)
    for (int j=0; j<N; j++) {
      integer ind = i*N+j;
      params[ind].i=i;
      params[ind].j=j;
      params[ind].pid=thread_create(f_wrapper,ind);
    }
  for (int k=0; k<N*N; k++) {
    wait(params[k].pid);
    cout<<params[k].r<<' ';
  }
}
```